# EtherNet/InterNet Driver AVRco NetStack *xUDP* preliminary

There are many ways how two or more units (processors, control units etc) can communicate with each other. One of them is Ethernet. The term „embedded Ethernet“ is heavily in use in the last time in the context with fast connections between control units or the connection between PC and control units.
Also the term „embedded TCP/IP“ is also in use with context. But one must be careful. TCP/IP is etablished as a common synonym for Ethernet. In reality with most cases not the TCP-protocol is implemented but an UDP/IP connection (eg. ProfiBUS-NET). To be honest, the TCP protocol sometimes is also implemented. But a sharp look to the internas shows that realtime jobs are always processed with UDP/IP. TCP/IP is not applicable for fast control functions because it uses a difficult and time consuming handshake protocol.
A network specialist now can argue that the UDP-protocol is very fast but also very unsecure. That is correct. UDP doesn't have any handshake. The sender transmits a packet and that's all. Whether the packet is received from the target or if there was an error, he can't know it because UDP doesn't have any response from the receiver build in.
This shortage of the „unsecure“ UDP-protocol is avoided by E-LAB (and also from most of the industrial UDP implementations) throu the use of an additional handshake implemented into the standard UDP protocol.
E-LAB calls this ***extended UDP*** or abreviated ***xUDP***.

# *xUDP*

With ***xUDP*** the **Sender** appends a 2Byte checksum to the data field. The checksum is calculated by an addition of all data bytes. The data block (byte count) of th senders is then incremented by 2Bytes. The **Receiver** now calculates it's own checksum from the data block exclusive the appended checksum. This checksum now is used as an acknowledge (handshake) for the reception of the packet. The receiver sends this checksum back to the sender (target = sender.UDPAknPort).

Now the **Sender** knows that the receivers at least has received the packet. He compares the received checksum with it's own, calculated at send-time. If these two numbers are equal the data packet was received by the receiver without any errors.

On the other side the **Receiver** compares it's own calculated checksum with this one, which was appended to the data packet. Now he also can ascertain whether the received packet is complete and error free.
With this implementation the „unsecure“ UDP is converted to the secure ***xUDP.***

With ***xUDP*** it's possible to implement a high speed communication which is more powerful for realtime applications as the TCP type. For these application which need TCP, FTP etc. E-LAB will provide a so called ***GateWay*** program (Redirector) which translates xUDP packets with the help of any PC connected to the same network into all other protocol types. This Gateway method is also used by other embedded Ethernet applications where resource useage/consumption is a problem (RAM, ROM) (eg. Motorola HC11/HC16).

Of course we know some TCP/IP and HTTP implementations also in a small PIC. But this can only be toys at all. They are unpracticable in the real world. If one really needs the complete Ethernet protocols in an embedded system he must ask himself whether he uses the wrong processor family ☺

## *Perfomance*

With a mega103 running at 6.4MHz the following transfer rates can be achieved :

PING 65 Bytes ca. 3msec PING 1450 Bytes ca. 10msec
Data transfer ca. 110kBytes/sec corresponds to a bitrate of ca. 1.0Mbit/sec

Code size ca. 6kBytes memory useage RAM ca. 400Bytes + RxBuffer + TxBuffer
Memory useage stack ca. 40 Bytes frame ca. 60 Bytes EEprom ca. 55 Bytes

## Import

As usual with AVRco the driver must be imported:

```
Import SysTick, NetStack, ..;
```

Because the generic hardware driver *NetStack_IOS* is not ready at this time, the internal hardware driver *RTL8019* must be imported. If the StackMapper PC-program is involved into the communication the additional used protocol(s) must be imported.

```
From NetStack Import RTL8019, xEmail, xFTP, xHTTP, xTelNet, xChatMode,
                     XFileMode, xPeerMode;

Define
        ProcClock    = 6400000;              // 6.4 MHz
        SysTick      = 10;                   // 10msec
        XData        = $8000,$FFFF;          // external data area
        XData1       = $4300,$4FFF, noInit;  // I/O area of RTL8019
        StackSize    = $30, iData;           // at least 50 bytes stacksize
        FrameSize    = $40, iData;           // at least 64 bytes frame size
        NetStack     = 1536, xData;          // total buffer size, location
        ARPcacheSize= 20;                    // arpcache entries
        RTL8019      = $4300, Int5;          // RTL base addr, Interruptpin
        RTLreset     = PortD, 2;             // reset line to RTL8019
```

## Exported types and constants

The NetStack driver exports several type declarations which can be or must be used by the application:

```
Type tMACaddress        = array[0..5] od byte;

Type tIPaddress         = array[0..3] od byte;

Type tGetPacketInfo     = record
                            TypeOfPacket : TPacketType;
                            SourceIP     : TIPAddress;
                            UserPortID   : byte;
                            SourcePort   : word;
                            DestPort     : Word;
                            DataBeginPtr : Pointer;
                            DataLength   : word;
                          End;

Type tUDPpackInfo       = record
                            TargetIPAddr: TIPAddress;
                            SourcePort  : Word;
                            DestPort    : Word;
                            WaitResp    : boolean;
                            DataPtr     : pointer;
                            DataLen     : word;
                          End;

Type tNetErrorType      = (NetErrNone, NetErrChkSum, NetErrSndPkt,
                           NetErrNoAkn, NetErrPktSize);

Type tPacketType        = (ptARP, ptPINGRequest, ptPINGResponse, ptICMP,
                           ptTCP, ptUDP, ptHTTP, ptFTP, ptTFTP, ptTELNET,
                           ptNONE);

Type tIdentType          = (idxRequMapper, idxFTP, idxEmail, idxTelNet,
                           idxChat, idxHTTP, idxFile);
```

```
type
  TIdentInfo   = record
                   IdentCode    : String[13];
                   BName        : String[20];
                   AknPort      : Word;
                   BufSize      : Word;
                   Stackversion: Word;
                   CompileDate  : String[10];
                   CompileTime  : String[5];
                   CompilerRev  : Word;
                   Firmware     : String[20];
                   xUDP         : boolean;
                 end;

  TMAPerror    = (MapErrNone, MapErrStackUnknown, MapErrFailed);

  TFTPcmd      = (FTPChangeDir, FTPChangeDirUp, FTPDelete, FTPMakeDir,
                  FTPRemoveDir, FTPRename, FTPSendFile, FTPFileAppend,
                  FTPReceiveFile);

  TFTPInfo     = record
                   Username      : string[20];
                   Password      : string[20];
                   FTPServer     : string[15];
                   Port          : word;
                   Filename      : string[32];
                   Filename1     : string[32];
                   TransfBinary  : boolean;
                   CmdCode       : tFTPcmd;
                   Result        : tMapError;
                 end;

  tMailInfo    = record
                   MailServer     : string[16];
                   Port           : word;
                   Sender         : string[32];
                   CC             : string[32];
                   BCC            : string[32];
                   UserID         : string[16];
                   ReplyTo        : string[16];
                   Subject        : string[16];
                   Result         : tMapError;
                 end;

  TFILEInfo    = record
                   Filename      : string[32];
                   Filename1     : string[32];
                   FILEcmd       : tFTPcmd;
                   Result        : tMapError;
                 end;

const
  StackIdentPort    = 61611;
  StackMapperPort   = 61612;
```

## *Exported variables*

There are many variables in use and also exported, but the applicated should not directly access them.
If a StackMapper Message has been received the kind of the message can be identified with the enumeration

```
Var MapperPackType : tIdentType;
```

Dependant of the content of the variable „MapperPackType“ the following records can be accessed:

```
Var RxMailInfo   : tMailInfo;
    RxFTPInfo    : tFTPInfo;
    RxTelNetInfo : tTelNetInfo;
    RxHTTPInfo   : tHTTPInfo;
    RxChatInfo   : tChatInfo;
    RxFileInfo   : tFileInfo;
    RxPeerInfo   : tPeerInfo;
```

If a message or command must be passed to the StackMapper one of the following provided variables must be used dependant of the message type:

```
Var TxMailInfo   : tMailInfo;
    TxFTPInfo    : tFTPInfo;
    TxTelNetInfo : tTelNetInfo;
    TxHTTPInfo   : tHTTPInfo;
    TxChatInfo   : tChatInfo;
    TxFileInfo   : tFileInfo;
    TxPeerInfo   : tPeerInfo;
```

## *Exported functions and procedures*

**Setup**

The following functions and procedures are normally only needed at program start.

```
Procedure Net_SetLocalIPAddress(IPOct4, IPOct3, IPOct2, IPOct1 : Byte;
                                MaskOct4, MaskOct3, MaskOct2, MaskOct1 : Byte);
```

Because the local IP-address must be placed into the eeprom, this procedure is only necessary if the local IP-address must be changed at runtime.

```
Procedure Net_SetDefaultGateway(Gateway4, Gateway3, Gateway2, Gateway1 :
                                Byte);
```

Also the Gateway address must be placed into the eeprom. So this procedure is only necessary if the Gateway address must be changed at runtime..

```
Procedure Net_StackSetAktive(YESNO : Boolean);
```

The NetStack driver is inactive by default. With this procedure the driver can be disconnected (false) and connected (true) to the network.

```
Procedure Net_ExtendedUDP(YESNO : Boolean);
```

The extended UDP mode is inactive by default. With this procedure the extended UDP-mode can be cleared (false) and enabled (true).

```
Function Net_AddUsrPort(Port : Word; ID : byte) : boolean;
```

With Net_AddUsrPort a port can be enabled which can be used now to receive packets. Port defines a port number and ID any value but which must be unique. These ID is is returned by the function `Net_GetPacket` in the record `GetPacketInfo` for information of the application.

```
Function Net_DelUsrPort(Port : Word) : boolean;
```

With Net_DelUsrPort an existing port is deleted from the Port LookUp table and therefore is never more accessible from outside.

**Runtime switches**

With these procedures the behaviour of the NetStacks for incoming requests can be controlled.

```
Procedure Net_AllowARPResponse(YESNO : Boolean);
```

A false parameter disables the scanning of the stack. This means that it doesn’t respond to a broadcast and therefore it is invisible. This is important for security applications. The switch is true by default.

**Procedure** Net_AllowPINGResponse(YESNO : Boolean);
A false disables pinging of the Stack. This means that an incoming PING will be ignored and the stack is invisible. This is important for security applications. The switch is true by default.

**Procedure** Net_AllowRequestBoard(YESNO : Boolean);
A false disables requesting internal data of the NetStack. This means that a BoardRequest (see below.) will be ignored. This is important for security applications. The switch is true by default.

**Operations**
These functions/procedures serve for the communication between the NetStack and other participiants of the network.

**Procedure** Net_StartNet;
These procedure starts the communication of the NetStack with the network.

**Function** Net_PollPacket : boolean;
These function returns the receive state of the NetStack. The result is true if a packet is present, otherwise it is false.

**Function** Net_GetPacket(**VAR** GetPacketInfo : TGetPacketInfo) : Boolean;
This function returns the record *GetPacketInfo* which contains all relevant informations of the received packet, on condition the boolean result is true. If the result is false, a packet has been received but either it was invalid or it has been processed for internal reasons like ARP, PING, NetStackInfo etc.
This function expects that a packet has been received. If this function is called without the knowledge that a packet is available, it loops until a packet has been received and then it returns. Because of this the function **Net_PollPacket** should be called at first. If it returns with a TRUE then Net_GetPacket must be called.

**Function** Net_SendUDPPacket(UDPinfo : tUDPpackInfo) : TNetErrorType;
This is the transmit function. The application passes all relevant information of the packet to send in the record *UDPInfo*. The result is of type tNetErrorType.

**Function** Net_RequestMapper(const IdentType : tIdentType) : TNetErrorType;
This function serves for registration and communication with a StackMapper/xUDPMapper. This is an E-LAB PC program which maps xUDP packets into FTP, E-MAIL etc. packets and transmits and receives packets from outside. The parameter „*IdentType*“ defines the general behaviour of this function.

| | |
|---|---|
| IdxRequMapper | the NetStack registrates itself at the Mapper. Registration is mandatory for all further Operations with the StackMapper. No additional data necessary and not possible. |
| IdxFTP | The NetStack contacts the Mapper for a pseudo-FTP connection/transfer. Therefore the record **TxFTPInfo** must be populated with the suiting parameters. |
| IdxEmail | The NetStack contacts the Mapper for a pseudo-EMail connection/transfer. Therefore the record **TxMailInfo** must be populated with the suiting parameters. |
| idxTelNet | The NetStack contacts the Mapper for a pseudo-TelNet connection/transfer. Therefore the record **TxTelNetInfo** must be populated with the suiting parameters. |
| idxChat | The NetStack contacts the Mapper for a ChatMode connection/transfer. Therefore the record **TxChatInfo** must be populated with the suiting parameters. |
| IdxHTTP | The NetStack contacts the Mapper for a pseudo-HTTP connection/transfer. Therefore the record **TxHTTPInfo** must be populated with the suiting parameters. |
| IdxFile | The NetStack initiates file-operations through the Mapper on the PC of the Mapper. Therfore the record **TxFileInfo** must be populated with the suiting parameters. |
| IdxPeer | The NetStack initiates file-operations through the Mapper on the Peer-to-Peer network. Therfore the record **TxPeerInfo** must be populated with the suiting parameters. |

Except of the IdxRequMapper call, a data packet can be appended to the info record which will be send by this function. The corresponding functions are „Net_TxInitDataBlock“ and „Net_TxAddDataStr“.
The result of the function is a „NetErrNone“ if the Mapper was found and the communication was successful. Otherwise a TimeOut error has occured. Except of „IdxReqMapper“ the StackMapper replies always with an xUDP reply packet which must be analized by the application. This info record is of the same type as the info send. The last entry contains a result info.

**Function** Net_TxInitDatablock(const IdentType : tIdentType) : pointer;
The function Net_RequestMapper can be instructed to append additional data to the info-record, for example the body of the e-mail message. Therefore the data block must be initialized before it data can be added. This must be done with this function. Also in case of an empty data block the call of this function is mandatory in order to avoid unexpected behaviour or errors.

**Function** Net_TxAddDataStr(st : string[32]) : boolean;
After the initialization of the data block with „Net_TxInitDataBlock" strings can be appended to the data block. The result „Pointer" of the above init function is not needed here. A success returns a true.

It's also possible to use the copy function „CopyBlock" to fill the data block. Example:

```
destPtr:= Net_TxInitDatablock;
CopyBlock(srcPtr, destPtr, count);
_TX_DATA_COUNT:= count;
err:= Net_RequestMapper(idxFTP);
```

The system variable _Tx_Data_Count must be set to the correct value, if CopyBlock or similar is used.

**Support functions**

**Procedure** STRtoIP(IPstr : String[15]; **VAR** Result : TIPAddress);
This function converts an IP-string with the format 'nn.nn.nn.nn' into an IP-address.

**Function** IPtoSTR(IP : TIPAddress) : String[15];
This function converts an IP-address into an IP-string with the format 'nn.nn.nn.nn'.

**Support tools**

At this time there are 3 support programs for test purposes and operation of the NetStacks.

**xUDPconf** is a program which can be called within the IDE PED32 and also operated separately. It serves for first time run of a NetStack hardware/software. The absolutely necessary setup of the board name, the IP-address and the MAC-Address can be done with this program.

**StackCheck** is a complete and superior test program where a entire network and all involved computers and NetStacks can be tested and checked thoroughly. It must be downloaded and installed separate. A user-ID for working is necessary.

**xUDPmapper** is a program which serves several jobs. The main job is to translate/map the xUDP packets from the local NetStacks into TCP/IP packets (FTP, http etc) and vice versa. It works as a link from NetStacks to the TCP/IP world. Because of this it should be placed into the autostart of one PC in the local network. The program *xUDPconf* is also build in. In addition a few simple but effective test function are build in. xUDPconf and xUDPmapper are always installed with the AVRco system. StackCheck must be installed separate.

## *Application*

The application must provide several constants for a valid behaviour of the NetStack.

```
const                              // into Flash
  // these 2 constants !must! be provided by the user
  Date = 'dd.mm.yyyy';
  Time = 'hh:mm';

{$EEPROM}
StructConst                       // into EEprom
  // first 2 bytes in AVR EEprom not useable
  eDummy             : word        = 0;
  // these 9 constants !must! be provided by the user
  Net_ARPTimeOut     : Byte        = 100;              // Systicks !!!
  Net_ARPRetry       : Byte        =  3;               // Request retries
  Net_SendTimeOut    : Byte        = 50;               // Systicks !!!
  Net_SendRetry      : Byte        = 5;                // Send retries
  Net_UDPAknPort     : Word        = 1189;             // Handshake port
  Net_LocalMACaddr   : tMACaddress = ($02, $2C, $2C, $CE, $AE, $15);
  Net_LocalIPaddr    : tIPaddress  = (192, 168, 1, 15);
  Net_LocalMask      : tIPaddress  = (255, 255, 255, 0);
  Net_xUDPMapper     : tIPaddress  = (0, 0, 0, 0);
  Net_DefaultGateWay: tIPaddress  = (0, 0, 0, 0);
  Net_BoardName      : String[20]  = 'E-LAB xUDP NET-Stack';
  Net_FirmWare       : String[20]  = 'E-LAB AVRco Test';
```

**Options**

```
procedure NetStack_LEDs(LED : byte; OnOff : boolean);
```

This optional CallBack procedure can be provided by the application. If present, the NetStack driver calls it for switching some LEDs on/off.

## *Details*

Not ready at this time.

```
Define
  XData
  XData1
  StackSize
  FrameSize
  NetStack
  ARPcacheSize
  RTL8019
  RTLreset

MACaddress
Ipaddress

GetPacketInfo
   TypeOfPacket
   SourceIP
   UserPortID
   SourcePort
   DestPort
   DataBeginPtr
   DataLength

UDPpackInfo
   TargetIPAddr
   SourcePort
```

```
   DestPort
   WaitResp
   DataPtr
   DataLen

NetErrorType
   NetErrNone
   NetErrChkSum
   NetErrSndPkt,
   NetErrNoAkn
   NetErrPktSize

PacketType
   PtARP
   PtPINGRequest
   PtPINGResponse
   ptICMP
   ptTCP
   ptUDP
   ptHTTP
   ptFTP
   ptTFTP
   ptTELNET
   ptNONE

Net_ARPTimeOut
Net_ARPRetry
Net_SendTimeOut
Net_SendRetry
Net_UDPAknPort
Net_LocalMACaddr
Net_LocalIPaddr
Net_LocalMask
Net_ReDirector
Net_DefaultGateWay
Net_BoardName
Net_FirmWare

StackIdentPort

TIdentInfo
   IdentCode
   BName
   AknPort
   BufSize
   Stackversion
   CompileDate
   CompileTime
   CompilerRev
   Firmware
   xUDP
```

**Example and schematics**

Several example program can be found in the installation directory **..\AVRco\Demos\NetWork**. The correspondig schematic **EtherBoardsch.PDF** is located in the directory **..\AVRco\DOCs**